NON-METALLIC HUMAN VAGUS NERVE STIMULATOR

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Mengqiao Li, Sijian Yan

MEng Field Advisor: Bruce Land

MEng Outside Advisor: Adam Anderson

Degree Date: August, 2016 / May, 2016

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Non-Metallic Human Vagus Nerve Stimulator

Author: Mengqiao Li, Sijian Yan

Abstract:

This project aims to create a human vagus nerve stimulator by using mechanical vibration. Users can stimulate the vagus nerve under different stimulation intensity by adjusting the frequency through a variable resister and a 555 Timer IC. The project consists of two parts: hardware implementation and experimental design. The hardware is basically the setup for the experiment. Results were concluded based on the data analysis after a great number of experiment was done. Therefore, the results have a high reliability and could be used for further development and application.

Individual Contribution

Sijian Yan

1. **A problem to be solved**

   The problems need to be solved by me in this project are: Using MATLAB for data analysis and Black Box Encapsulation. Furthermore, Mengqiao and I did the vibration generator circuit testing and heart rate data collection together.

2. **A review of possible options for solution**

   2.1 Analyze Data by MATLAB

   - Pan-Tompkins Algorithm
   - Develop my own algorithm

   2.2 Black Box Encapsulation

   I didn't think of any other options besides encapsulate everything into a black box.

3. **What formulates the "best" solution**

   At last, I chose to use Pan-Tompkins Algorithm since it is a famous and rigorous algorithm, the results should be more reliable. Moreover, EKG signal is too complicated to be analyzed correctly and completely. Therefore, as a starter in biomedical research, I decided to refer Pan-Tompkins and make some revise instead of coming up with my own algorithm.

4. **Documentation of design implementation**

   4.1 Analyze data by Pan-Tompkins Algorithm

   The data which was analyzed by Pan-Tompkins Algorithm is the data transformed by a piece of python program which was originally recorded by BioPac device.

   4.2 Black Box Encapsulation

   First of all, I fixed the circuit board and the power supply unit in the bottom of the box. Then, I drilled three holes on the lid of the box for potentiometer, switch, and signal output respectively. Last, I attached the potentiometer and switch onto the lid, and led the output out of the box through a thick wire which was sorted to the ear phone.

5. Testing of the final results with regard to the original specifications

5.1 Results of Pan-Tompkins Algorithm

The Pan-Tompkins Algorithm works well. Peak location figures were generated correctly after the signal went through the algorithm in MATLAB

5.2 Results of Black Box Encapsulation

The Black Box works well. Users can turn on/off the ear phone’s vibration by the switch fixed on the box. Furthermore, vibration frequency can be changed by turning the potentiometer on the box.

Individual Contribution

Mengqiao Li

1. **Data processing**

   Due to the reason that there were multiple outputs from Pan-Tompkins Algorithm in MATLAB and were not directly analyzable, Li wrote a program to extract and preprocess the data of interest.

2. **Data analysis**

   Li created a program that achieved the following functions:

   a. generating beat the beat intervals

   b. calculating mean intervals under different states and frequencies

   c. calculating standard deviation of intervals

3. **Result visualization**

   In order to make it comfortable for researchers to notice the conclusion from the results, Li created graphs and charts to display the comparisons between different frequencies and different states such as density distribution and standard deviation.

4. **Circuit board building**

   Li built the circuit board with Yan.

5. **Circuit board testing**

   Li tested the circuit board with Yan and made necessary adjustments.

6. **Data collecting experiment**

   Li collected raw heart rate data with Yan by conducting experiments using biological device—BioPac MP150 and software—Psychopy.

Executive Summary

This design project is created to find a noninvasive way to stimulate human vagus nerve. The noninvasive way is achieved by using mechanical vibration. Detailed implementation will involve a variable frequency generator design, cardiovascular measurements, data analysis by R, JAVA and MATLAB. The final goal aimed to fulfill is to find out some correlation between stimulating vagus nerve by mechanical vibration attached to left ear and a more variable heart beat rate. Furthermore, this invention can also be used for medical research since vagus nerve stimulation is most often used to treat depression and epilepsy when other treatments haven't worked.

The non-metallic human vagus nerve stimulator project contains of two parts: implementing a vibration generator and doing heart rate data analysis.

The vibration generator is consisted of:

- 555 Timer IC – an integrated chip used as an electronic oscillator to generate pulse
- Potentiometer – a three-terminal resister with a rotating contact that can be used as a voltage divider to change the frequency of 555 Timer IC
- Power supply – two AAA batteries controlled by a 2-terminal switch
- Motor – a 3VDC vibration little pancake motor
- Earphone – a plastic, in-ear earphone used to conduct the vibration generated by motor

The heart rate data analysis is consisted of:

- Pan-Tompkins algorithm – is a real-time algorithm used for QRS complexes detection of ECG/EKG signals
- Interval distribution – a way to observe heart rate varying extent
- Standard deviation – a measure of how spread out intervals are
- Chi-square testing - a statistical hypothesis test to judge if the proposed hypothesis is true

In conclusion, the vibration generator reached all of the expected functionality; the heart rate data analysis was done correctly by programing. However, the overall results did not match the expectation due to too few experiment subjects. Therefore, further test is needed.

Table of Contents

Table of Figures

1 Introduction

Vagus nerve is the longest cranial nerve in human's body, which connects to heart, esophagus, lungs, and so on. Because of such many organs it passes by, vagus nerve has significant impacts in a lot of ways; for example, keeping the heart rate constant and controlling food digestion.

Nowadays, the most common ways of vagus nerve stimulation is electrical stimulation – implanting a device under the skin on one's chest that connects to the left vagus nerve with a wire. When the device is activated, it sends electrical signals along the vagus nerve to certain areas in brain which will have a positive impact on the treatment of epilepsy or depression. However, the electrical stimulation is a kind of invasive method so it may cause extra pain for the patient due to the surgery.

Our research project wants to test a potential hack by utilizing the fact that the Vagus receives input from touch receptors around the ear. Both of Tragus and Cavity of concha contain 45% of vagus nerve receptors. Antihelix contains 75% and all of the receptors in Cymba Conchac are vagus nerve receptor. These preconditions make the ear vagus nerve stimulation become realizable. Therefore, this project will involve designing an around-the-ear device that uses mechanical vibrations of various frequencies. This will be interfaced with existing cardiovascular measurements to assess what frequencies of stimulation may enhance vagal outflow. On the software side, Pan-Tomkins algorithm was revised and used in peaks locating in MATLAB. Python, JAVA, and R are also used in statistical data analysis.

2 System Design

2.1 High-Level Design

Figure 1 demonstrates the high-level system design concept. We inserted a 3V DC Vibration Motor, which is triggered by 555 Timer IC and powered by the AAA power supply, into the earphone. Then put the earphone in the tester's left ear. By regulating the potentiometer, users are able to change the vibration frequency which further change the tester's heart beat frequency.

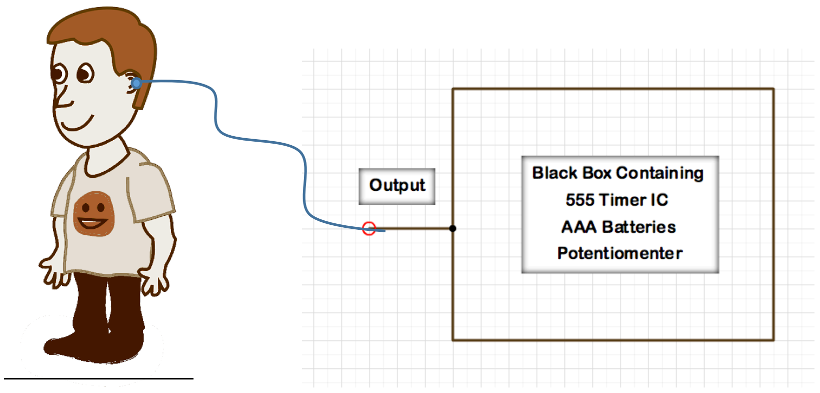


Figure 1 High-level Design

Figure 2 shows the flowchart of the whole system. One set of 2 AAA batteries is the power supply of the circuit board and motor. EKG device is plugged into the 120V on the wall. The tester's heart beat (raw data) is recorded by EKG device while his/her vagus nerve in left or right ear is stimulated by the motor. A piece of python grogram is used to convert the time interval and amplitude of the heart beat from electronic signal to numerical numbers, which are further analyzed by MATLAB, JAVA and R to get more intuitively results.

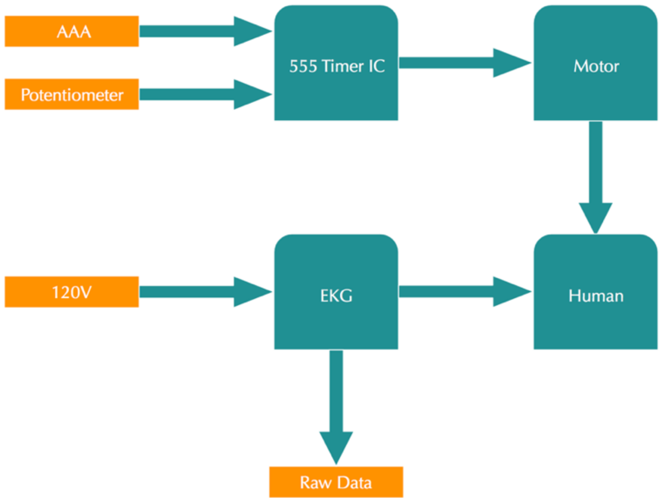


Figure 2 System Flowchart

2.2 Hardware Design

The following subsections describe hardware design and materials selection.

2.2.1 555 Timer IC

555 Timer IC is an integrated chip which can be used to produce accurate time delays, generate pulse, and oscillate. In this project, it plays as an oscillator to generate variable output frequency. Figure 4 and Figure 5 are photo and pin configuration of the most common 555 Timer IC.

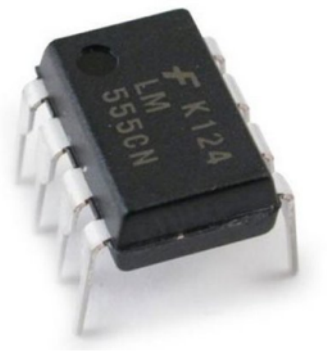


Figure 3 555 Timer IC

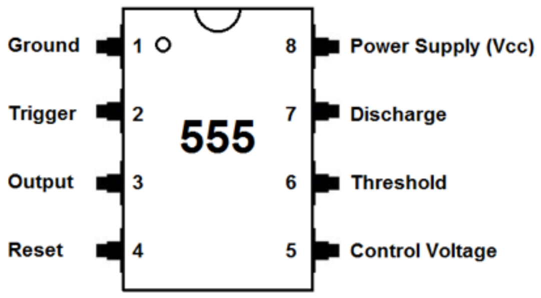


Figure 4 Pin Configuration of 555 Timer IC

555 Timer IC specification	
Model Number	LM555
Operating Voltage	4.5V – 16.0V
Threshold Voltage	3.33V – 10.0V
Physical Dimension	9.46mm * 6.35mm
Rise / Fall Time	100 ns

Table 1 Specification of 555 Timer IC

2.2.2 Potentiometer

A potentiometer in Figure 7 is a three-terminal 15kΩ resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor, which is the function applied in this project. Basically, a potentiometer consists of five parts: a resistive element, a sliding contact, the wiper, that moves along the element, electrical terminals at each end of the element, a mechanism that moves the wiper from one end to the other, and a housing containing the element and wiper. Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. In the project, it is used for adjusting the output frequencies.

Figure 5 Potentiometer

2.2.3 Power Supply

Besides the 120V induced from the wall, the other power supply used in the project is a set of triple-A batteries. An AAA battery in Figure 8 is a standard size of dry cell battery commonly used in low drain portable electronic devices, which provides 1.5V each.

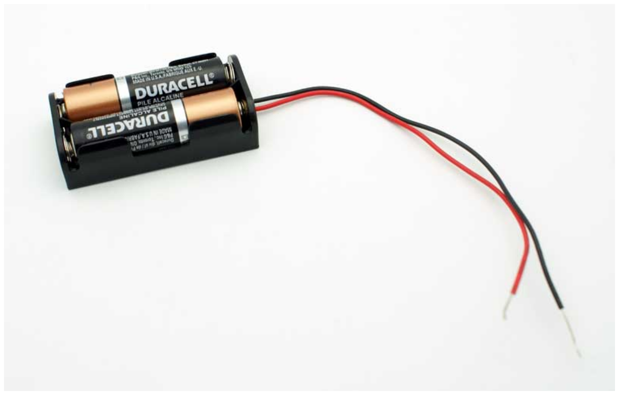


Figure 6 AAA Batteries

2.2.4 BioPac MP150

BioPac MP150 is a 16-Channel data acquisition & analysis system which could communicate with computer by Ethernet. It is used for acquiring heart beat signal from EDA/PPG Transponder; then it transports the signal to laptop where the signal is digitalized by a piece of python program for statistical analysis.

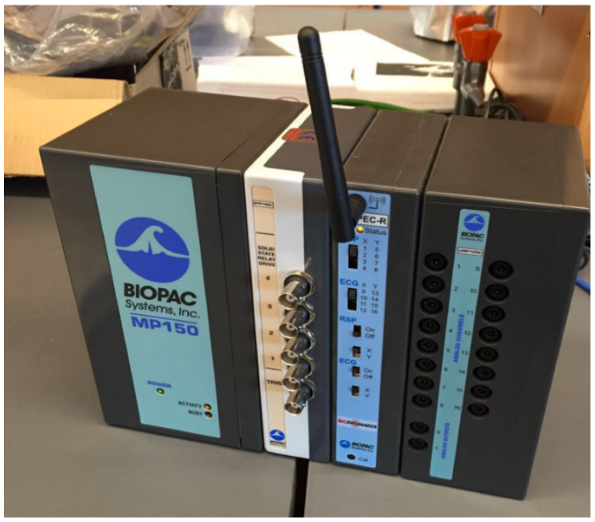


Figure 7 BioPac MP150

2.3 Software Design

2.3.1 Heart Rate Data Collection

Given the BioPac device, software is still needed in order to get the digital data of heart rate information. By running the python scripts provided by researchers from Human Ecology department, the wave of heart rate can be observed and it automatically generated a .csv file containing all the values throughout the time.

2.3.2 Analyzable Data Extraction and Preprocessing

Holding the heart rate wave values throughout time, we first needed to find the wave-peak positions to proceed further analysis. Pan-Tompkins algorithm perfectly meets the need and there was existing MATLAB Pan-Tompkins toolbox online. The algorithm was applied; the positions of all the wave peaks were extracted by using MATLAB.

Pan-Tompkins algorithm is a real-time algorithm commonly used for detecting QRS complexes, a combination of three different graphical reflections in a EKG plot, of ECG/EKG signals, which was came up by Jiapu Pan and Willis J. Tompkins in 1985.

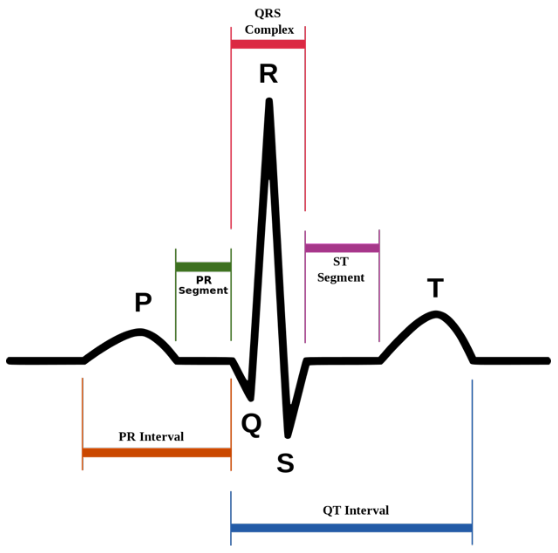


Figure 8 QRS Complex

The signal goes through a low pass filter, a high pass filter, and a bandpass filter to get rid of noise in the MATLAB program. Then the signal is differentiated to provide the QRS complex slope information.

Low pass filter transfer function: $\boldsymbol{H(z)} = \frac{(1-z^{-6})^2}{(1-z^{-1})^2}$

High pass filter transfer function: $\boldsymbol{H(z)} = \frac{-1+32z^{-16}+z^{-32}}{1+z^{-1}}$

Derivative filter transfer function: $\boldsymbol{H(z)} = \frac{-z^{-2}-2z^{-1}+2z+z^{2}}{8T}$ (T = sampling period)

After differentiation, the signal is squared point by point which makes all data points positive and also nonlinearly enhance the dominant peaks.

Squaring function: $\boldsymbol{y(nT)} = [\boldsymbol{x(nT)}]^2$.

Next, moving-window integration is executed to obtain waveform feature information in addition to the slope of the R wave.

Moving-window integration: $\boldsymbol{y(nT)} = \frac{x(nT-(N-1)T)+x(nT-(N-2)T)+\cdots+x(nT)}{N}$ (N = the number of samples in the width of the integration window)

It is physiologically impossible for R wave to occur in less than 200ms distance; therefore, a minimum distance of 40 samples is considered between each R wave, which can be used to determine the fiducial mark. Finally, thresholds need to be adjusted in order to locate the peaks, which are local maximums determined by when the signal changes direction within a predefined time interval.

Finally, in order to utilize the position data in different integrated developments environment (IDE), a Java program was created to preprocess the data generated from MATLAB to make it analyzable.

2.3.3 Statistical Analysis

Firstly, a Java program was written which achieved the following functions:

- Calculate peak-to-peak intervals using the analyzable position information we got from data preprocessing
- Compute mean heart beat intervals in different situations
- Compute standard deviations of the interval distribution

Then, Chi-square test is executed to further analyze the correlations between different tested conditions.

Chi-square test is a kind of statistical hypothesis test in which the sampling distribution of the test is a chi-square distribution when the proposed hypothesis is true. It is applied by computing the square errors of the sample: $\frac{(observed-expected)^2}{expected}$, then compare the square errors to the value of the Chi-square distribution under different degree of freedom (DF).

2.3.4 Data Visualization

In order to make it easy to observe and compare, several R scripts were created to display density distribution and standard deviation.

2.4 Experimental Design

2.4.1 Group Dividing

Vagus nerve is considered to be existing on the left side of human's ear, so both sides needed to be stimulated in order to compare. Apart from experiments on left and right ears, the heart rate when neither side of ear is stimulated is also needed to be tested as a control group to accurately get the results. Control group is necessary since testers might be affected even if the stimulator is not directly touching their skin. Besides, different frequencies needed to be used to stimulate in order to find the one that most significantly influence heart rate.

These facts lead to two variables in this experiment: states and frequencies. Based on the precondition, the experiment was divided into 3 groups:

- Control: with the stimulator vibrating in the face of subjects but not touching the ear
- Left ear: with the stimulator vibrating in the left ear
- Right ear: with the stimulator vibrating in the right ear

Within each group, 3 different frequencies were implemented:

- 10Hz
- 30Hz
- 60Hz

2.4.2 Repeated Measurement

Repeated measurement is a common way to generate reliable data. Due to the grouping method, 9 testing groups were executed in total which are:

Groups	**Without stimulation**	**Left ear stimulation**	**Right ear stimulation**
10 Hz	w/o, 10 Hz	left, 10 Hz	right, 10 Hz
30 Hz	w/o, 30 Hz	left, 30 Hz	right, 30 Hz
60 Hz	w/o, 60 Hz	left, 60 Hz	right, 60 Hz

Table 2 Testing Groups

For each group, the heart rate was recorded in 1 minute and the step was repeated for 3 times. Therefore, 27 (= 1 min * 3 times * 9 groups) groups of heart rate data were collected at last.

3 Implementation and Experimental Results

3.1 Implementation

3.1.1 555 Timer IC Integration

In the 555 Timer IC integration circuit in Figure 9, pin 2 and pin 6 are connected together allowing the circuit to re-trigger itself on each cycle. Every cycle allows it to operate as a free running oscillator. During each cycle, capacitor C0 charges up through both timing resistors, 15kΩ potentiometer R1 and 1kΩ register R2, but discharges itself only through resistor R2, as the other side of R2 is connected to the discharge terminal, pin 7.

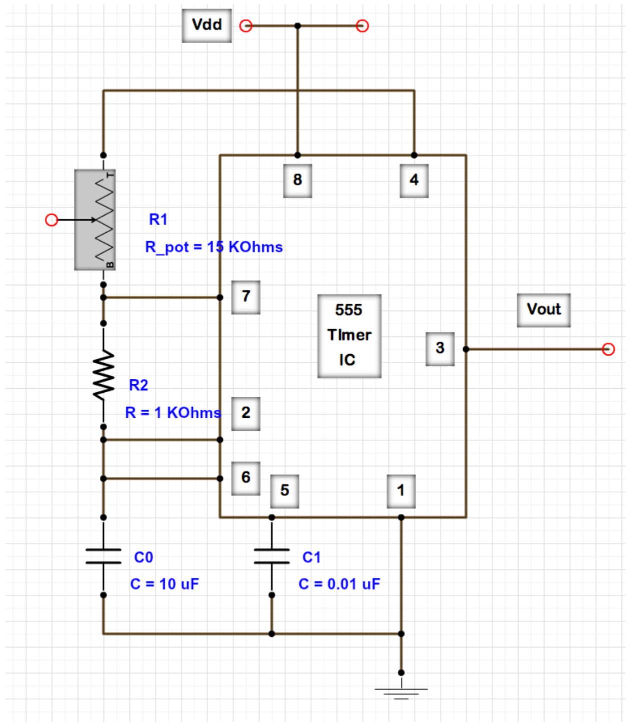


Figure 9 555 Timer IC Integration Schematic

3.1.2 Black Box Implementation

The black box is constructed by a solderable breadboard with necessary electronic components, a potentiometer, a set of AAA batteries, and a switch. Turning the switch on enables the ear phone to vibrate; turning the switch off stops vibration. Turning the potentiometer towards “down” decreases the vibration frequency and intensity; on the contrary, turning it towards “up” increases the vibration frequency and intensity.

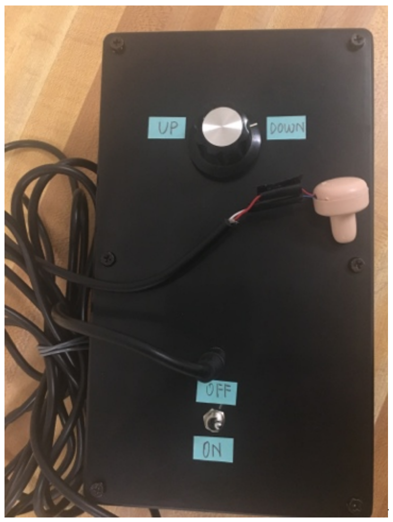


Figure 10 Black Box

3.2 Experimental Results

3.2.1 Heart Beat Wave

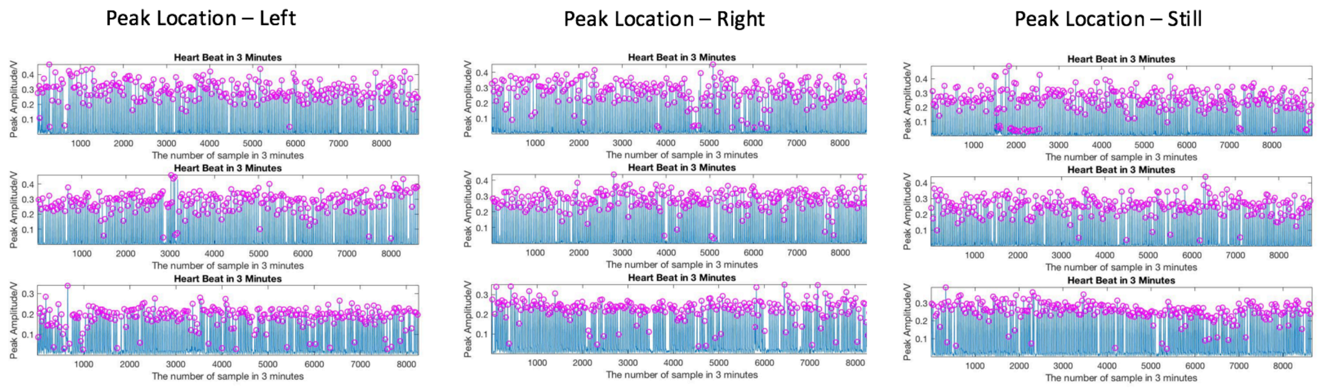


Figure 11 Peak-location

Figure 11 shows the heart beat amplitude (from single subject) versus time. Each X-axis represents three minutes and the plots from top to bottom are measured under 10Hz, 30Hz, and 60Hz stimulation respectively.

Heart beat waves themselves cannot really indicate any features. Heart beat intervals were then computed to observe more characteristics of stimulation.

3.2.2 Heart Beat Intervals

Figure 12 represents the heart beat interval distribution under different stimulation frequencies. Normally the wider the peak is, the more variable the heart rate is considered to be. It's easy to notice that heart rate varies the most when using 10 Hz frequency without the stimulator directly touching the ear. 60 Hz frequency appears to be the same among groups no matter vagus nerve is being stimulated or not. Heart rate varies less under 10Hz frequency when it is stimulated than when it is not stimulated. Whereas it varies more under 30Hz frequency when it is stimulated.

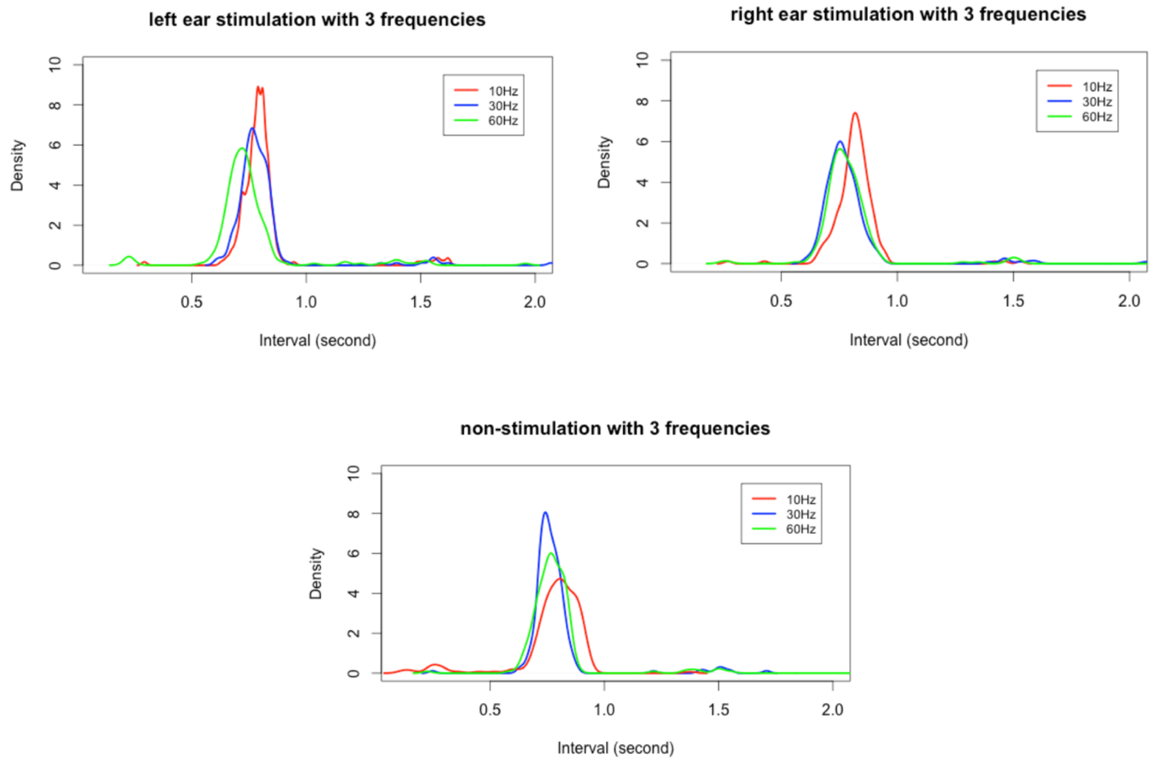


Figure 12 Interval Distribution

4 Conclusion

4.1 Evaluation

4.1.1 Standard deviation

Greater standard deviation of beat-to-beat intervals indicates a more varying heart rate. Figure 13 indicates that heart rate varies the most when stimulating right ear under frequency at 60Hz. However, it violates the hypothesis that stimulating the vagus nerve in left ear would vary the heart beat most.

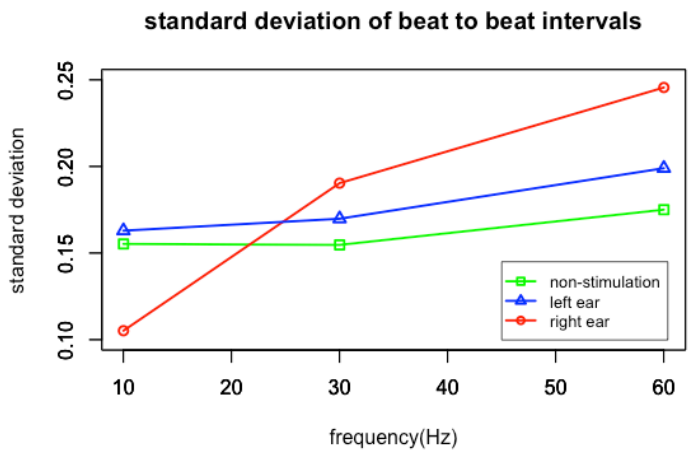


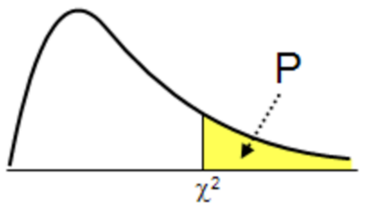
Figure 13 Standard Deviation

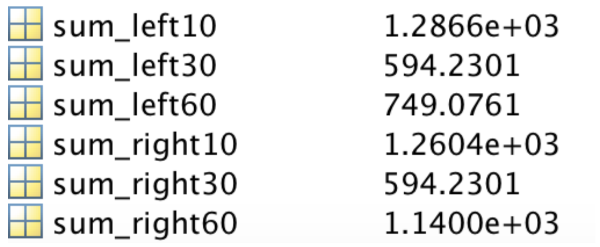
4.1.2 Chi-square Test

In order to evaluate the correlations of stimulating left ear V.S. still and stimulation right ear V.S. still, Chi-squared test was constructed to determine whether there is a significant difference between the expected frequencies and the observed frequencies in the experiments.

In Figure 14, the variables start with "sum" represent the summation of $\frac{(left-still)^2}{still}$ and $\frac{(right-still)^2}{still}$, which are the square errors in each group. In the experiment, the degree of freedom is 226 since the number of sample is 227. All of the numbers exceed the standard value in the table above. Therefore, it is not convincing to conclude any relationship based on Chi-Square calculation. Further test is needed.

Values of the Chi-squared distribution





sum_left10	1.2866e+03
sum_left30	594.2301
sum_left60	749.0761
sum_right10	1.2604e+03
sum_right30	594.2301
sum_right60	1.1400e+03

	P										
DF	**0.995**	**0.975**	**0.20**	**0.10**	**0.05**	**0.025**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	0.0000393	0.000982	1.642	2.706	3.841	5.024	5.412	6.635	7.879	9.550	10.828
2	0.0100	0.0506	3.219	4.605	5.991	7.378	7.824	9.210	10.597	12.429	13.816

......

225	174.116	185.348	242.631	252.578	260.992	268.438	270.681	277.269	283.390	290.925	296.288
226	174.995	186.256	243.671	253.638	262.070	269.530	271.777	278.379	284.511	292.061	297.433
227	175.874	187.164	244.711	254.699	263.147	270.622	272.874	279.488	285.632	293.196	298.579

Figure 14 Chi-square Test

4.2 Future Improvement

Currently this vagus nerve stimulator has achieved fundamental functions. Statistical analysis of experimental results using this stimulator doesn't indicate a more varying heart rate under vagus nerve stimulating circumstances so further evaluations are needed. It is probably caused by the limited number of testers—which is just one in these experiments.

Further work could focus on making a better shaped, more comfortable end of the device, as well as implementing an MRI compatible capsule. It could be a 25-feet tubing with a speaker on one end which is metallic placing outside the MRI environment and an earphone-like 3D printed end on the other side in the MRI environment, which is inserted in tester's ear. As for experiments, more testing subjects are needed in order to get more reliable results.

5 Acknowledgement

We would like to thank our advisor Dr. Bruce Land for his advice, encouragement, and continued support of this project.

We also need to thank to Dr. Adam Anderson and his PhD student Ross Markello for their external help with using lab equipment and teaching biomedical knowledge.

6 References

[1] Wikipedia. (2016) "Vagus Nerve." [Online]. Available: https://en.wikipedia.org/wiki/Vagus_nerve

[2] MAYO CLINIC. (2015) "Vagus Nerve Stimulation." [Online]. Available: http://www.mayoclinic.org/tests-procedures/vagus-nerve-stimulation/home/ovc-20167755

[3] Fairchild Semiconductor. (2013). "LM555 Single Timer." [Online]. Available: https://www.fairchildsemi.com/datasheets/LM/LM555.pdf

[4] Wikipedia. (2016). "Potentiometer". [Online]. Available: https://en.wikipedia.org/wiki/Potentiometer

[5] BioPac. (2016). "BIOPAC MP150: THE FASTEST, EASIEST WAY TO BETTER DATA". [Online]. Available: http://www.biopac.com/product-category/research/systems/mp150-starter-systems/

[6] IEEE. (1985). "A Real-Time QRS Detection Algorithm". [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4122029

[7] MathWorks. (2014). "Complete Pan Tompkins Implementation ECG QRS Detector" [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/45840-complete-pan-tompkins-implementation-ecg-qrs-detector/content/pan_tompkin.m

[8] Wikipedia. (2016). "Chi-squared test." [Online]. Available: https://en.wikipedia.org/wiki/Chi-squared_test

[Figure 3] Circuits Today. (2014). "555 Timer – A Complete Basic Guide." [Online]. Available: http://www.circuitstoday.com/555-timer

[Figure 4] Learning about Electronics. (2014). "555 Timer Pinout." [Online]. Avaliable: http://www.learningaboutelectronics.com/Articles/555-timer-pinout.php

[Figure 5] Digi-Key. (2016). "Honeywell-Sensing-and-Productivity-Solutions 53C11MEGZ". [Online]. Available: http://www.digikey.com/product-detail/en/honeywell-sensing-and-productivity-solutions/53C11MEGZ/480-6572-ND/5037507

[Figure 6] Vetco. (2013). "Battery Powered Ardunio – AA, AAA, 9V". [Online]. Available: http://www.vetco.net/blog/?p=113

[Figure 8] Wikipedia. (2016). "QRS complex." [Online]. Available: https://en.wikipedia.org/wiki/QRS_complex

[Figure 14] MadCalc. (2016). “Values of the Chi-squared distribution”. [Online]. Available: https://www.medcalc.org/manual/chi-square-table.php

7 Appendix

7.1 Appendix A. Wave

```
#Authored by: Ross M.
#Created: 12/07/2015

# # # # # # # #
# IMPORTS

from __future__ import division
from psychopy import visual, event
import numpy as np
import random, os


# # # # # # # #
# CONSTANTS

DUMMY_RECORDING = False
WINDOW_SIZE = (800,800)
SAMPLE_RATE = 1000
LOG_FILE = 'testlog'

if not DUMMY_RECORDING:
    from libmpdev import MP150


# # # # # # # #
# SET UP STIM

# display
win = visual.Window(size=WINDOW_SIZE,units='norm')

# shape for waveform (starts at center of screen)
waveForm = visual.ShapeStim(win,closeShape=False,vertices=[[0,0],[0,0]])

# text for baseline measurement
baseText = visual.TextStim(win,wrapWidth=2,text=
         "Please take five deep breaths, inhaling and exhaling "
         "completely each time. Please refrain from holding your "+
         "breath, if at all possible."
         "\n\n"+
         "Press space after you have taken five breaths.")

# start communication with MP150
if not DUMMY_RECORDING:
    mp = MP150(logfile=LOG_FILE,samplerate=SAMPLE_RATE)


# # # # # # # #
# EXPERIMENT

if not DUMMY_RECORDING:
    #start recording measurements to log file
    mp.start_recording()

    # make a baseline measurement of at least one breath for normalization
    baseline = []
    # instructions for baseline breath
    baseText.draw()
    win.flip()

    while len(event.getKeys(keyList='space')) == 0:
        baseline.append(mp.sample()[0])

    win.flip(clearBuffer=True)

    # normalizing function for future breathing
    normalizer = max(abs(np.max(baseline)),abs(np.min(baseline)))

# now, display a constantly updating waveform on the screen
# press escape to exit
while len(event.getKeys(keyList='escape')) == 0:
    # grab a sample from MP 150
    if not DUMMY_RECORDING:
        sample = mp.sample()[0]
        currPoint = (0,sample/normalizer)
    # or make a random sample
    else:
        t = random.uniform(100,1000)
        x = random.uniform(0,100)
        mod = random.sample([-1,1],1)[0]
        currPoint = (0,mod*(x/t))

    numVertices = waveForm.vertices.shape[0]

    # move the waveform along the screen until it extends to the left edge
    if numVertices < 100:
        times, points = np.split(waveForm.vertices,2,axis=1)
        times = (np.arange(-1*numVertices,1)/100).reshape(numVertices+1,1)
        points = np.vstack((points,currPoint[1]))
        waveForm.vertices = np.hstack((times,points))
    else:
        times, points = np.split(waveForm.vertices,2,axis=1)
        points = np.vstack((np.split(points,[1],axis=0)[1],currPoint[1]))
        waveForm.vertices = np.hstack((times,points))

    # draw the line to the screen and refresh!
    waveForm.draw()
    win.flip()


# # # # # # # #
# SHUT IT DOWN

if not DUMMY_RECORDING:
    mp.stop_recording()
    mp.close()

win.close()
```

7.2 Appendix B. Pan-Tompkins

```
function [qrs_amp_raw,qrs_i_raw,delay]=pan_tompkin(ecg,fs,gr)
```

```
%% Author : Hooman Sedghamiz
% Linkoping university

% Any direct or indirect use of this code should be referenced
% Copyright march 2014

if ~isvector(ecg)
  error('ecg must be a row or column vector');
end

if nargin < 3
    gr = 1;   % on default the function always plots
end
ecg = ecg(:); % vectorize

%% Initialize
qrs_c =[]; %amplitude of R
qrs_i =[]; %index
SIG_LEV = 0;
nois_c =[];
nois_i =[];
delay = 0;
skip = 0; % becomes one when a T wave is detected
not_nois = 0; % it is not noise when not_nois = 1
selected_RR =[]; % Selected RR intervals
m_selected_RR = 0;
mean_RR = 0;
qrs_i_raw =[];
qrs_amp_raw=[];
ser_back = 0;
test_m = 0;
SIGL_buf = [];
NOISL_buf = [];
THRS_buf = [];
SIGL_buf1 = [];
NOISL_buf1 = [];
THRS_buf1 = [];

%% Noise cancelation(Filtering) % Filters (Filter in between 5-15 Hz)
if fs == 200

%% Low Pass Filter  H(z) = ((1 - z^(-6))^2)/(1 - z^(-1))^2
b = [1 0 0 0 0 0 -2 0 0 0 0 0 1];
a = [1 -2 1];
h_l = filter(b,a,[1 zeros(1,12)]);
ecg_l = conv (ecg ,h_l);
ecg_l = ecg_l/ max( abs(ecg_l));
delay = 6; %based on the paper

%% High Pass filter H(z) = (-1+32z^(-16)+z^(-32))/(1+z^(-1))
b = [-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 32 -32 0 0 0 0 0 0 0 0 0 0 0 0 0 1];
a = [1 -1];
h_h = filter(b,a,[1 zeros(1,32)]);
ecg_h = conv (ecg_l ,h_h);
ecg_h = ecg_h/ max( abs(ecg_h));
delay = delay + 16; % 16 samples for highpass filtering
else
```

```
%% bandpass filter for Noise cancelation of other sampling
frequencies(Filtering)
f1=5; %cuttoff low frequency to get rid of baseline wander
f2=15; %cuttoff frequency to discard high frequency noise
Wn=[f1 f2]*2/fs; % cutt off based on fs
N = 3; % order of 3 less processing
[a,b] = butter(N,Wn); %bandpass filtering
ecg_h = filtfilt(a,b,ecg);
ecg_h = ecg_h/ max( abs(ecg_h));
end

%% derivative filter H(z) = (1/8T)(-z^(-2) - 2z^(-1) + 2z + z^(2))
h_d = [-1 -2 0 2 1]*(1/8);%1/8*fs
ecg_d = conv (ecg_h ,h_d);
ecg_d = ecg_d/max(ecg_d);
delay = delay + 2; % delay of derivative filter 2 samples

%% Squaring nonlinearly enhance the dominant peaks
ecg_s = ecg_d.^2;

%% Moving average Y(nt) = (1/N)[x(nT-(N - 1)T)+ x(nT - (N - 2)T)+...+x(nT)]
ecg_m = conv(ecg_s ,ones(1 ,round(0.150*fs))/round(0.150*fs));
delay = delay + 15;

if gr
plot(ecg_m);axis tight;title('Heart Beat in 3 Minutes');
axis tight;
xlabel('The number of sample in 3 minutes') ;
ylabel('Peak Amplitude/V');
end

%% Fiducial Mark
% Note : a minimum distance of 40 samples is considered between each R wave
% since in physiological point of view no RR wave can occur in less than
% 200 msec distance
[pks,locs] = findpeaks(ecg_m,'MINPEAKDISTANCE',round(0.2*fs));

%% initialize the training phase (2 seconds of the signal) to determine the
THR_SIG and THR_NOISE
THR_SIG = max(ecg_m(1:2*fs))*1/3; % 0.25 of the max amplitude
THR_NOISE = mean(ecg_m(1:2*fs))*1/2; % 0.5 of the mean signal is considered
to be noise
SIG_LEV= THR_SIG;
NOISE_LEV = THR_NOISE;

%% Initialize bandpath filter threshold(2 seconds of the bandpass signal)
THR_SIG1 = max(ecg_h(1:2*fs))*1/3; % 0.25 of the max amplitude
THR_NOISE1 = mean(ecg_h(1:2*fs))*1/2; %
SIG_LEV1 = THR_SIG1; % Signal level in Bandpassed filter
NOISE_LEV1 = THR_NOISE1; % Noise level in Bandpassed filter

%% Thresholding and online desicion rule

for i = 1 : length(pks)
```

```
   %% locate the corresponding peak in the filtered signal
    if locs(i)-round(0.150*fs)>= 1 && locs(i)<= length(ecg_h)
          [y_i x_i] = max(ecg_h(locs(i)-round(0.150*fs):locs(i)));
       else
          if i == 1
            [y_i x_i] = max(ecg_h(1:locs(i)));
            ser_back = 1;
          elseif locs(i)>= length(ecg_h)
            [y_i x_i] = max(ecg_h(locs(i)-round(0.150*fs):end));
          end

     end

  %% update the heart_rate (Two heart rate means one the moste recent and the
other selected)
    if length(qrs_c) >= 9

        diffRR = diff(qrs_i(end-8:end)); %calculate RR interval
        mean_RR = mean(diffRR); % calculate the mean of 8 previous R waves
interval
        comp =qrs_i(end)-qrs_i(end-1); %latest RR
        if comp <= 0.92*mean_RR || comp >= 1.16*mean_RR
            % lower down thresholds to detect better in MVI
                THR_SIG = 0.5*(THR_SIG);
                %THR_NOISE = 0.5*(THR_SIG);
               % lower down thresholds to detect better in Bandpass filtered
                THR_SIG1 = 0.5*(THR_SIG1);
                %THR_NOISE1 = 0.5*(THR_SIG1);

        else
            m_selected_RR = mean_RR; %the latest regular beats mean
        end

    end

      %% calculate the mean of the last 8 R waves to make sure that QRS is
not
       % missing(If no R detected , trigger a search back) 1.66*mean

       if m_selected_RR
           test_m = m_selected_RR; %if the regular RR availabe use it
       elseif mean_RR && m_selected_RR == 0
           test_m = mean_RR;
       else
           test_m = 0;
       end

    if test_m
          if (locs(i) - qrs_i(end)) >= round(1.66*test_m)% it shows a QRS is
missed
              [pks_temp,locs_temp] = max(ecg_m(qrs_i(end)+
round(0.200*fs):locs(i)-round(0.200*fs))); % search back and locate the max
in this interval
              locs_temp = qrs_i(end)+ round(0.200*fs) + locs_temp -
1; %location
```

```
               if pks_temp > THR_NOISE
                qrs_c = [qrs_c pks_temp];
                qrs_i = [qrs_i locs_temp];

                % find the location in filtered sig
                if locs_temp <= length(ecg_h)
                 [y_i_t x_i_t] = max(ecg_h(locs_temp-
round(0.150*fs):locs_temp));
                else
                 [y_i_t x_i_t] = max(ecg_h(locs_temp-round(0.150*fs):end));
                end
                % take care of bandpass signal threshold
                if y_i_t > THR_NOISE1

                       qrs_i_raw = [qrs_i_raw locs_temp-round(0.150*fs)+
(x_i_t - 1)];% save index of bandpass
                       qrs_amp_raw =[qrs_amp_raw y_i_t]; %save amplitude of
bandpass
                       SIG_LEV1 = 0.25*y_i_t + 0.75*SIG_LEV1; %when found with
the second thres
                end

                not_nois = 1;
                SIG_LEV = 0.25*pks_temp + 0.75*SIG_LEV ;  %when found with the
second threshold
              end

           else
               not_nois = 0;

           end
    end

    %%  find noise and QRS peaks
    if pks(i) >= THR_SIG

                 % if a QRS candidate occurs within 360ms of the previous QRS
                 % ,the algorithm determines if its T wave or QRS
                 if length(qrs_c) >= 3
                      if (locs(i)-qrs_i(end)) <= round(0.3600*fs)
                        Slope1 = mean(diff(ecg_m(locs(i)-
round(0.075*fs):locs(i)))); %mean slope of the waveform at that position
                        Slope2 = mean(diff(ecg_m(qrs_i(end)-
round(0.075*fs):qrs_i(end)))); %mean slope of previous R wave
                             if abs(Slope1) <= abs(0.5*(Slope2))  % slope
less then 0.5 of previous R
                                 nois_c = [nois_c pks(i)];
                                 nois_i = [nois_i locs(i)];
                                 skip = 1; % T wave identification
                                 % adjust noise level in both filtered and
                                 % MVI
                                 NOISE_LEV1 = 0.125*y_i + 0.875*NOISE_LEV1;
                                 NOISE_LEV = 0.125*pks(i) + 0.875*NOISE_LEV;
                             else
                                 skip = 0;
                             end
```

```
                    end
               end

        if skip == 0  % skip is 1 when a T wave is detected
        qrs_c = [qrs_c pks(i)];
        qrs_i = [qrs_i locs(i)];

        % bandpass filter check threshold
         if y_i >= THR_SIG1
                        if ser_back
                           qrs_i_raw = [qrs_i_raw x_i];  % save index of
bandpass
                        else
                           qrs_i_raw = [qrs_i_raw locs(i)-round(0.150*fs)+
(x_i - 1)];% save index of bandpass
                        end
                           qrs_amp_raw =[qrs_amp_raw y_i];% save amplitude of
bandpass
          SIG_LEV1 = 0.125*y_i + 0.875*SIG_LEV1;% adjust threshold for
bandpass filtered sig
         end

        % adjust Signal level
        SIG_LEV = 0.125*pks(i) + 0.875*SIG_LEV ;
        end

    elseif THR_NOISE <= pks(i) && pks(i)<THR_SIG

         %adjust Noise level in filtered sig
         NOISE_LEV1 = 0.125*y_i + 0.875*NOISE_LEV1;
         %adjust Noise level in MVI
         NOISE_LEV = 0.125*pks(i) + 0.875*NOISE_LEV;

    elseif pks(i) < THR_NOISE
        nois_c = [nois_c pks(i)];
        nois_i = [nois_i locs(i)];

        % noise level in filtered signal
        NOISE_LEV1 = 0.125*y_i + 0.875*NOISE_LEV1;
        %end

         %adjust Noise level in MVI
        NOISE_LEV = 0.125*pks(i) + 0.875*NOISE_LEV;

    end

    %% adjust the threshold with SNR
    if NOISE_LEV ~= 0 || SIG_LEV ~= 0
        THR_SIG = NOISE_LEV + 0.25*(abs(SIG_LEV - NOISE_LEV));
        THR_NOISE = 0.5*(THR_SIG);
    end

    % adjust the threshold with SNR for bandpassed signal
    if NOISE_LEV1 ~= 0 || SIG_LEV1 ~= 0
        THR_SIG1 = NOISE_LEV1 + 0.25*(abs(SIG_LEV1 - NOISE_LEV1));
        THR_NOISE1 = 0.5*(THR_SIG1);
    end

% take a track of thresholds of smoothed signal
SIGL_buf = [SIGL_buf SIG_LEV];
NOISL_buf = [NOISL_buf NOISE_LEV];
THRS_buf = [THRS_buf THR_SIG];

% take a track of thresholds of filtered signal
SIGL_buf1 = [SIGL_buf1 SIG_LEV1];
NOISL_buf1 = [NOISL_buf1 NOISE_LEV1];
THRS_buf1 = [THRS_buf1 THR_SIG1];

 skip = 0; %reset parameters
 not_nois = 0; %reset parameters
 ser_back = 0;  %reset bandpass param
end

if gr
hold on,scatter(qrs_i,qrs_c,'m');
    disp(qrs_i_raw);

end
end
```

7.3 Appendix C. Peaks Locating

```
filenameA = 'left_10';
columnA = xlsread(filenameA,'B2:B8847');
figure(1);
subplot(311);
pan_tompkin(columnA, 31, 1);

hold on;
filenameB = 'left_30';
columnB = xlsread(filenameB,'B2:B8752');
subplot(312);
pan_tompkin(columnB, 31, 1);

hold on;
filenameC = 'left_60';
columnC = xlsread(filenameC,'B2:B8248');
subplot(313);
pan_tompkin(columnC, 60, 1);

filenameD = 'right_10';
columnD = xlsread(filenameD,'B2:B8735');
figure(2);
subplot(311);
pan_tompkin(columnD, 31, 1);

hold on;
filenameE = 'right_30';
columnE = xlsread(filenameE,'B2:B8723');
subplot(312);
pan_tompkin(columnE, 31, 1);
```

```
hold on;
filenameF = 'right_60';
columnF = xlsread(filenameF,'B2:B8368');
subplot(313);
pan_tompkin(columnF, 60, 1);

filenameG = 'still_10';
columnG = xlsread(filenameG,'B2:B8928');
figure(3);
subplot(311);
pan_tompkin(columnG, 31, 1);

hold on;
filenameH = 'still_30';
columnH = xlsread(filenameH,'B2:B8752');
subplot(312);
pan_tompkin(columnH, 31, 1);

hold on;
filenameI = 'still_60';
columnI = xlsread(filenameI,'B2:B8644');
subplot(313);
pan_tompkin(columnI, 60, 1);
```

7.4 Appendix D. Interval Distribution and Standard Deviation

```
import java.util.Arrays;

public class SdOfIntervals {

      public static void main(String[] args) {
            String s60 ;
            String s30 ;
            String s10 ;
            String r60 ;
            String r30 ;
            String r10 ;
            String l60 ;
            String l30 ;
            String l10 ;

            String[] all = {s60,s30,s10,r60,r30,r10,l60,l30,l10};
            for(String s : all) {
                  int[] out = getIntervals(s);
                  float[] toSecond = new float[out.length];
//                System.out.println("================This is a new 
state===============");
                  for(int i = 0; i < out.length; i++) {
                        toSecond[i] = (float)out[i]*180/8000;
//                      System.out.print((i == out.length -1)? toSecond[i] : 
(toSecond[i] + ","));
                  }
//                System.out.println();
//                System.out.println(sdArray(toSecond));
                  System.out.println(mean(toSecond));
            }
            for(String s : all) {
                  int[] out = getIntervals(s);
                  float[] toSecond = new float[out.length];
//                System.out.println("================This is a new
state===============");
                  for(int i = 0; i < out.length; i++) {
                        toSecond[i] = (float)out[i]*180/8000;
//                      System.out.print((i == out.length -1)? toSecond[i] :
(toSecond[i] + ","));
                  }
//                System.out.println();
                  System.out.print(sdArray(toSecond) + ",");
//                System.out.println(mean(toSecond));
            }
      }

      /** March 22: normalize the measurements + return String version of
display method*/
      public static void gimmeString(String str) {
            int[] intervals = getIntervals(str);
            StringBuilder sb = new StringBuilder();
            for (float i : intervals) {
                  i = i * 60 /3300;
                  sb.append(i);
            }
            String gimme = sb.toString();
            display(gimme);
      }

      /** Take into a string and display it in a clear way
       * without the influences of all spaces */
      public static void display(String str) {
            int[] intervals = getIntervals(str);
            System.out.println();
            System.out.println(Arrays.toString(intervals));
      }

      public static int[] getIntervals(String s) {
            int[] nums = getNums(s);
            int size = nums.length-1;
            int[] intervals = new int[size];
            for(int i = 0; i < size; i++) {
                  intervals[i] = nums[i+1] - nums[i];
            }
            return intervals;
      }

      public static double sdArray(float[] intervals) {
            float size = intervals.length;
            float m = mean(intervals);
            float sum = 0;
            for(int i = 0; i < size; i++) {
                  sum += (intervals[i] - m) * (intervals[i] - m);
            }
            sum /= size;
            return Math.sqrt(sum);
      }

      private static int[] getNums(String s) {
            String[] sArray = s.split("\\s+");
            int[] numbers = new int[sArray.length];
```

```
        for(int i = 0; i < sArray.length; i++) {
            numbers[i] = Integer.parseInt(sArray[i]);
        }
        //System.out.println(sArray.length);
        return numbers;
    }

    private static float mean(float[] nums) {
        float size = nums.length;
        float total = 0;
        for(int i = 0; i < size; i++) {
            total += nums[i];
        }
        return total/size;
    }
}
```

7.5 Appendix E. Interval Distribution Visualization

```
dens_l10<-density(l10)
dens_l30<-density(l30)
dens_l60<-density(l60)
dens_r10<-density(r10)
dens_r30<-density(r30)
dens_r60<-density(r60)
dens_s10<-density(s10)
dens_s30<-density(s30)
dens_s60<-density(s60)

plot(dens_l10, col="red",xlim=c(0.1,2),ylim=c(0,10),lwd=2.5, main='left ear
 stimulation with 3 frequencies', xlab="Interval (second)")
lines(dens_l30, col="blue",lwd=2.5)
lines(dens_l60,col="green",lwd=2.5)
legend(1.6,9.5,legend=c("10Hz","30Hz","60Hz"),lwd=2.5,col=c("red",
 "blue","green"), lty=1:1, cex=0.8)

plot(dens_r10, col="red", xlim=c(0.1,2),ylim=c(0,10),lwd=2.5, main='right ear
 stimulation with 3 frequencies', xlab="Interval (second)")
lines(dens_r30, col="blue",lwd=2.5)
lines(dens_r60,col="green",lwd=2.5)
legend(1.6,9.5,legend=c("10Hz","30Hz","60Hz"),lwd=2.5,col=c("red",
 "blue","green"), lty=1:1, cex=0.8)

plot(dens_s30, col="blue",xlim=c(0.1,2),ylim=c(0,10),lwd=2.5, main='non-
 stimulation with 3 frequencies', xlab="Interval (second)")
lines(dens_s10, col="red", lwd=2.5)
lines(dens_s60,col="green",lwd=2.5)
legend(1.6,9.5,legend=c("10Hz","30Hz","60Hz"),lwd=2.5,col=c("red",
"blue","green"), lty=1:1, cex=0.8)
```

7.6 Appendix F. Standard Deviation Visualization

```
ssd<-c(0.17507343837486067,0.15471520031374503,0.15528768797925024)
rsd<-c(0.24560127482236768,0.19038364147140582,0.1051657173350168)
lsd<-c(0.19899398615750089,0.1698306650053749,0.16292568716782593)
hist(ssd)
x<-c(60,30,10)
y<-c(0.1,0.25)
plot.new()
```

```
plot(x,ssd,pch=0,col="green",type="o",ylim=y,ylab="standard
 deviation",xlab="frequency(Hz)",lwd=2.5,main='standard deviation of beat to
 beat intervals')
par(new=TRUE)
plot(x,rsd,pch=1,col="red",type="o",ylim=y,ylab="",xlab="",lwd=2.5)
par(new=TRUE)
plot(x,lsd,pch=2,col="blue",type="o",ylim=y,ylab="",xlab="",lwd=2.5)
legend(45,0.145,legend=c("non-stimulation","left ear","right
 ear"),lwd=2.5,col=c("green","blue","red"),pch=c(0,2,1),lty=1:1,cex=0.8)
```

7.7 Appendix G. Chi-square Test

```
filename_interval = 'interval';
column_still_60 = xlsread(filename_interval,'A2:A228');
column_still_30 = xlsread(filename_interval,'A230:A456');
column_still_10 = xlsread(filename_interval,'A458:A684');
column_right_60 = xlsread(filename_interval,'A686:A912');
column_right_30 = xlsread(filename_interval,'A914:A1140');
column_right_10 = xlsread(filename_interval,'A1142:A1368');
column_left_60 = xlsread(filename_interval,'A1370:A1596');
column_left_30 = xlsread(filename_interval,'A1598:A1824');
column_left_10 = xlsread(filename_interval,'A1826:A2052');

left10 = (column_left_10-column_still_10).^2;
chi_square_left10 = left10./column_still_10;
sum_left10=sum(chi_square_left10);

left30 = (column_left_30-column_still_30).^2;
chi_square_left30 = left30./column_still_30;
sum_left30=sum(chi_square_left30);

left60 = (column_left_60-column_still_60).^2;
chi_square_left60 = left60./column_still_60;
sum_left60=sum(chi_square_left60);

right10 = (column_right_10-column_still_10).^2;
chi_square_right10 = right10./column_still_10;
sum_right10=sum(chi_square_right10);

right30 = (column_right_30-column_still_30).^2;
chi_square_right30 = left30./column_still_30;
sum_right30=sum(chi_square_right30);

right60 = (column_right_60-column_still_60).^2;
chi_square_right60 = right60./column_still_60;
sum_right60=sum(chi_square_right60);

ave_left10=sum(column_left_10)/227;
ave_left30=sum(column_left_30)/227;
ave_left60=sum(column_left_60)/227;

ave_right10=sum(column_right_10)/227;
ave_right30=sum(column_right_30)/227;
ave_right60=sum(column_right_60)/227;

ave_still10=sum(column_still_10)/227;
```

```
ave_still30=sum(column_still_30)/227;
ave_still60=sum(column_still_60)/227;

SD_left10=sqrt(sum((column_left_10-ave_left10).^2)/227);
SD_left30=sqrt(sum((column_left_30-ave_left30).^2)/227);
SD_left60=sqrt(sum((column_left_60-ave_left60).^2)/227);

SD_right10=sqrt(sum((column_right_10-ave_right10).^2)/227);
SD_right30=sqrt(sum((column_right_30-ave_right30).^2)/227);
SD_right60=sqrt(sum((column_right_60-ave_right60).^2)/227);

SD_still10=sqrt(sum((column_still_10-ave_still10).^2)/227);
SD_still30=sqrt(sum((column_still_30-ave_still30).^2)/227);
SD_still60=sqrt(sum((column_still_60-ave_still60).^2)/227);
```